

Algoritmos y Estructuras de Datos II

Primer recuperatorio

Miércoles 7 de julio de 2021

Aclaraciones

- El examen es individual en todas sus etapas, y a libro abierto.
- Se deberán justificar todas las respuestas realizadas.
- Se responderán consultas por Discord, sólo de interpretación del enunciado, a través del canal ‘Mesa de Docentes’ de la categoría PARCIALES.
- Cualquier aclaración sobre el enunciado se realizará a través del canal ‘anuncios-evaluaciones’.
- El examen durará 5 horas, de 9 a 14hs, y la entrega se realizará a través del Campus hasta esa hora, en forma estricta. Se desestimarán entregas fuera de tiempo, sin excepciones.
- Cada ejercicio se calificará con **P**erfecto, **A**probado, **R**egular, o **I**nsuficiente.
- Para aprobar la cursada se deben tener todos los ejercicios aprobados de todos los parciales.

Ej. 1. Especificación: axiomatización

Extender el TAD Multiconjunto con las siguientes operaciones:

1. $\text{eliminarTodos} : \text{multiconj}(\alpha) \times \alpha \longrightarrow \text{multiconj}(\alpha)$

que dado un multiconjunto m y un elemento e , elimina todas las apariciones de e en m .

Por ejemplo: $\text{eliminarTodos}(\{1, 1, 2, 2, 2, 3, 3, 3, 4\}, 3) = \{1, 1, 2, 2, 2, 4\}$.

2. $\text{Nrepetidos} : \text{multiconj}(\alpha) \times \text{nat} \longrightarrow \text{conj}$

que dado un multiconjunto m y un número natural n , devuelve el conjunto de los elementos que aparecen repetidos **por lo menos** n veces en m .

Por ejemplo: $\text{Nrepetidos}(\{1, 1, 2, 2, 2, 3, 3, 3, 4\}, 2) = \{1, 2, 3\}$.

3. $\text{maxRepetidos} : \text{multiconj}(\alpha) \times m \longrightarrow \text{conj} \quad \{\neg \emptyset?(m)\}$

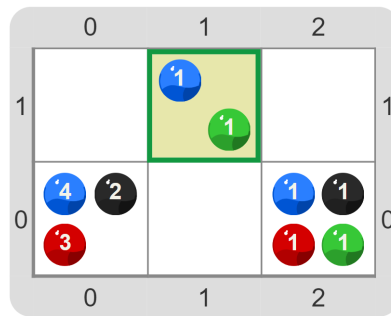
que dado un multiconjunto no vacío m , devuelve el conjunto de los elementos que aparecen repetidos más veces en m .

Por ejemplo: $\text{maxRepetidos}(\{1, 1, 2, 2, 2, 3, 3, 3, 4\}) = \{2, 3\}$.

Ej. 2. Especificación: modelado

En el lenguaje de programación Gobstones¹ se modela la memoria de una computadora con un tablero fijo de $N \times M$ celdas, donde cada celda puede contener una cantidad no negativa de bolitas de alguno de los cuatro colores: Rojo, Azul, Verde y Negro. Además, se cuenta con un cabezal de lectura/escritura que se puede ubicar en cualquiera de las celdas del tablero. Así, llamaremos *celda actual* a la celda en la cual está ubicado el cabezal.

En la siguiente imagen se muestra un ejemplo de un tablero de 2×3 con el cabezal ubicado en la posición $\langle 1, 1 \rangle$.



La celda actual tiene una bolita verde y una bolita azul. La celda en la posición $\langle 0, 0 \rangle$ tiene 2 bolitas negras, 3 rojas y 4 azules. La celda en la posición $\langle 0, 2 \rangle$ tiene una bolita de cada color. Las demás celdas están vacías.

Las instrucciones básicas del lenguaje son:

- **Mover**: dada una dirección, mueve el cabezal una posición en esa dirección.
- **Poner**: dado un color, pone una bolita de ese color en la celda actual.
- **nroBolitas**: dado un color, devuelve la cantidad de bolitas de ese color que hay en la celda actual.

En la versión original de Gobstones el tablero es de tamaño fijo y el cabezal no puede moverse más allá de los límites del mismo. Actualmente se está trabajando en un prototipo para hacer que el tamaño del tablero sea variable. Así, el tablero iniciará siempre con una única celda vacía y el cabezal sobre ella (es decir, las dimensiones del tablero inicial serán 1×1 y la ubicación del cabezal en dicho tablero será la posición $\langle 0, 0 \rangle$). Cada vez que el cabezal intente moverse sobre un borde del tablero, en lugar de fallar, deberá agregarse una fila o una columna, según corresponda para agrandar el tablero y que el cabezal pueda moverse. Tener en cuenta que las ubicaciones deben seguir siendo números no negativos así que tras agregar una columna sobre el borde Oeste o una fila sobre el borde Sur se deben actualizar, además de las dimensiones del tablero, las ubicaciones de todas las celdas.

Utilizando los siguientes TADs ya provistos, y suponiendo definida la suma entre tuplas de enteros, especificar el TAD **TABLERO** que permita describir el comportamiento deseado (de la nueva versión de Gobstones), teniendo en cuenta que además se desea saber cuál es la celda que tiene más bolitas en total.

Sugerencia: definir un observador **celdasConBolitas** que describa el conjunto de celdas **NO VACÍAS** del tablero.

La especificación debe estar completa, incluyendo tanto el modelo (igualdad observacional, observadores, generadores y otras operaciones con sus restricciones) como las axiomatizaciones.

TAD COLOR es {AZUL, ROJO, VERDE, NEGRO}
TAD DIRECCIÓN es {ESTE, OESTE, SUR, NORTE}
TAD POSICIÓN es TUPLA(NAT, NAT)

¹<https://gobstones.github.io/>

observadores básicos

$$\text{posición} : \text{celda} \longrightarrow \text{posición}$$
$$\text{vacía} : \text{posicion} \longrightarrow \text{celda}$$
$$\text{mover} : \text{celda} \times \text{direccion} \longrightarrow \text{celda}$$
$$\text{nroBolitas}(\text{vacía}(\text{p}), \text{k}) \quad \equiv \quad 0$$
$$\text{nroBolitas}(\text{poner}(c, k'), k) \quad \equiv \quad \text{nroBolitas}(c, k) + \beta(k = k')$$
$$\text{nroBolitas}(\text{mover}(c, d), k) \quad \equiv \quad \text{nroBolitas}(c, k)$$
$$\text{posición}(\text{vacía}(\text{p})) \equiv \text{p}$$
$$\text{posición}(\text{poner}(c, k)) \quad \equiv \quad \text{posición}(c)$$

```

posición(mover(c, d))
≡ posición(c) + if d = Este then
     $\langle 1, 0 \rangle$ 
else
    if d = Oeste then
         $\langle -1, 0 \rangle$ 
    else
        if d = Sur then  $\langle 0, -1 \rangle$  else  $\langle 0, 1 \rangle$  fi
    fi
fi

```

Fin TAD

Ej. 3. Complejidad: funciones

Sean a y b dos constantes enteras tales que $a > b \geq 1$. Sea $f : \mathbb{N} \rightarrow \mathbb{R}^+$ una función.

Determinar si las siguientes afirmaciones son verdaderas o falsas. Justificar debidamente.

1. $(n + a)^b \in \Theta((n + b)^a)$
2. $(n + a)! \in \Omega((n + b)!)$
3. $f(n)^a \in O(n^a)$
4. $f(n)^b \in O(f(n)^a)$
5. Si $f(n) \in \Theta(n^a)$ entonces $f(n) \in \Omega(n^b)$
6. Si $f(n) \in O(n^a)$ entonces $f(n)^2 \in O(n^{2a})$

Ej. 4. Complejidad: análisis de algoritmos

Una matriz cuadrada es *diagonal* si todos los elementos fuera de la diagonal principal son iguales a 0. Es decir, una matriz M de $n \times n$ es diagonal si $M[i][j] = 0$ para todos los $i \neq j$. Por ejemplo, las matrices A , B y C son diagonales, pero D no lo es.

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 8 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Nos dan la siguiente función para determinar si una matriz cuadrada es diagonal:

```
función MATRIZDIAGONAL(matriz  $M$ )
   $n := \text{LONG}(M[0])$ 
   $i := 0$ 
   $esDiagonal := \text{true}$ 
  mientras  $esDiagonal$  y  $i < n$  hacer
    para  $j := 0 \dots n - 1$  hacer
      si  $i \neq j$  y  $M[i][j] \neq 0$  entonces
        |  $esDiagonal := \text{false}$ 
      fin
    fin
     $i := i + 1$ 
  fin
  devolver  $esDiagonal$ 
fin
```

Supongamos que las operaciones para determinar la longitud de una fila de la matriz, para acceder a los elementos de la matriz, y para comparar los elementos de la matriz con 0, se calculan en tiempo constante.

1. ¿Cuál es la complejidad temporal del mejor caso de $\text{MATRIZDIAGONAL}(M)$? Justificar (describir una familia de entradas en las que se obtiene dicha complejidad, realizar el cálculo de complejidad en esos casos, y mostrar que no se puede obtener un mejor caso).
2. ¿Cuál es la complejidad temporal del peor caso de $\text{MATRIZDIAGONAL}(M)$? Justificar.

En ambos casos, dar la cota de complejidad más ajustada posible.

Ej. 5. Invariante de representación y función de abstracción

El Ministerio de Salud de la Tierra de Algo Dos (TAD) nos pide que especifiquemos un sistema para administrar sus vacunas contra el Coronavirus SARS-CoV-2. En cada ciudad de este país se cuenta con un centro de vacunación para inocular a sus habitantes. Como la vacunación es voluntaria, las personas interesadas deben inscribirse previamente para recibir la vacuna. Además, sólo se pueden vacunar en la ciudad en la que residen, ya que una persona tiene domicilio en una única ciudad.

La vacuna es de tipo monodosis, esto es, de una única dosis por persona, mientras que los frascos de vacunas vienen con 5 dosis cada uno, y sólo se puede abrir uno si hay al menos 5 personas disponibles para vacunar. Las dosis de vacunas van llegando a los centros de las distintas ciudades en distintos momentos, y de distintas cantidades. Las inscripciones van llegando a los respectivos centros en forma de planilla con una o varias personas anotadas. Las personas que ya recibieron su dosis no pueden volver a anotarse en la planilla. En cada centro, la administración de vacunas se realiza inmediatamente siempre que haya personas inscriptas esperando y vacunas disponibles, pero sin violar la anterior restricción de los frascos. Las personas a vacunar serán escogidas por el sistema de acuerdo a un criterio que se definirá más adelante, en la etapa de diseño.

Nos pidieron especificar con un TAD el Sistema de Vacunación (SV) descripto, teniendo en cuenta que además se desea saber cuál fue la ciudad que más vacunó y cuál tiene más vacunas disponibles sin usar (en todos los casos, de haber más de una ciudad que cumpla con lo indicado, se elige alguna sin ninguna preferencia puntual). Para ello, la cátedra propuso la siguiente solución:

TAD CIUDAD es STRING

TAD PERSONA es NAT

TAD SV

...

observadores básicos

ciudades : sv \rightarrow conj(ciudad)

residentes : sv $s \times$ ciudad $c \rightarrow$ conj(persona) $\{c \in \text{ciudades}(s)\}$

vacunados : sv $s \times$ ciudad $c \rightarrow$ conj(persona) $\{c \in \text{ciudades}(s)\}$

esperando : sv $s \times$ ciudad $c \rightarrow$ conj(persona) $\{c \in \text{ciudades}(s)\}$

#frascosPorCiudad : sv $s \times$ ciudad $c \rightarrow$ nat $\{c \in \text{ciudades}(s)\}$

generadores

iniciarSV : dicc(persona, ciudad) $d \rightarrow$ sv $\{\neg \emptyset?(claves(d))\}$

traerFrascos : sv $s \times$ ciudad $c \times$ nat $n \rightarrow$ sv $\{c \in \text{ciudades}(s) \wedge n > 0\}$

inscribirPersonas : sv $s \times$ ciudad $c \times$ conj(persona) $ps \rightarrow$ sv $\left\{ \begin{array}{l} c \in \text{ciudades}(s) \wedge (\forall p : \text{persona})(p \in ps \Rightarrow p \in \text{residentes}(s, c) \wedge \neg(p \in \text{esperando}(s, c)) \wedge \neg(p \in \text{vacunados}(s, c))) \end{array} \right\}$

otras operaciones

ciudadQueMásVacunó : sv \rightarrow ciudad

ciudadConMásFrascos : sv \rightarrow ciudad

...

axiomas

...

Fin TAD

Ahora queremos representarlo mediante la siguiente estructura:

sv **se representa con** estr, donde

estr es tupla

```
<  vacunados  :  dicc(ciudad, secu(persona))
    esperando  :  dicc(ciudad, conj(persona))
    frascos    :  conj(tupla(ciudad, nat))
    residencia :  dicc(persona, ciudad)
>
```

Informalmente, esta representación cumple las siguientes propiedades:

- En *vacunados* están todas las personas vacunadas por ciudad, en algún orden.
- En *esperando* guardamos, para cada ciudad del país, todas las personas que están esperando su vacuna allí.
- En *frascos* indicamos cuántos frascos disponibles hay por ciudad.
- En *residencia* tenemos la información de dónde vive cada persona del país.

Se pide:

1. Escribir en forma coloquial el invariante de representación.
2. Escribir formalmente el invariante de representación.
3. Escribir formalmente la función de abstracción.